MODIS
SCIENCE DATA SUPPORT TEAM
PRESENTATION

October 2, 1992

AGENDA

Page

1. Action Items 1
2. MODIS Airborne Simulator (MAS) 2
3. MODIS Level-1 Software Design 9
4. MODIS Level-2 Processing Shell 12
5. MODIS Level-1 Earth Navigation 18

ACTION ITEMS:

06/12/92 [Tom Goff, Carroll Hood] Develop separate detailed schedules using Microsoft Project for Level-1A and -1B software design and development. (Updated results for Level-1B are included in the handout.) STATUS: Open. Due Date: 07/10/92

07/31/92 [Tom Goff, Ed Masuoka, Al Fleig] Develop the purpose and requirements for a packet simulator. Get more information on the packet simulator being developed by SBRC. (An updated requirements specification was included in the handout on 09/04/92. A copy, with a cover letter, should be sent Jerry Hyde of SBRC for coordination with their requirements.) STATUS: Open. Due Date: 09/04/92

MODIS Airborne Simulator (MAS) Status

Liam E. Gumley
Progress up to 1 October 1992

(1) Software/data developments

I worked with Si-Chee Tsay on getting him up to speed with NetCDF over the last few weeks. One concern he expressed with the MAS Level-1B datasets was that an estimate of the exoatmospheric solar spectral irradiance in each channel was not included. His comment was that the visible/near-infrared channels were "useless" without this information. Most users will need this information to do any quantitative analysis of the information in the visible/near-IR channels. In the interim I supplied Si-Chee with a program I extracted from the LOWTRAN7 code that returns the solar irradiance at a given wavelength. LOWTRAN7 has a database of solar spectral irradiance at the mean Earth-Sun distance from 1.74 microns to 500 microns at resolution of 10 or 20 wavenumbers. I propose to compute an Earth-orbit corrected solar spectral irradiance for each MAS channel weighted by the spectral response of that channel i.e.

$$S_W(\lambda) = \frac{\int S(\lambda)\, R(\lambda)\, d\lambda}{\int R(\lambda)\, d\lambda}$$

where $S_W(\lambda)$ = sensor weighted orbit corrected exoatmospheric solar spectral irradiance
$S(\lambda)$ = orbit corrected exoatmospheric solar spectral irradiance
$R(\lambda)$ = sensor spectral response

I already have all of the software components necessary to compute these numbers.

If this information is included in MAS Level-1B datasets, it may also be advisable to include information which allows users to convert from radiance in the infrared channels to temperature. This is a common step in quantitative analysis of infrared radiance data. Temperature to radiance conversion tables are already computed as part of the calibration processing, and it would be a simple matter to include these as part of the Level-1B datasets. These tables contain the sensor weighted Planck radiances at 1K intervals from 150K to 373K for each infrared channel. If these were included in the Level-1B files then users could perform interpolation to go from a given radiance value to the corresponding temperature value. This information would add about 10000 bytes to the size of each Level-1B output file (usually tens to hundreds of megabytes in total size).

(2) Noise computations

I received a set of noise computation results from Chris Moeller at Wisconsin that I compared to my own estimates. These are presented overleaf.

Channel	Wavelength (microns)	SNR(1)	SNR(2)	Temp(1) (Kelvin)	Temp(2) (Kelvin)
7	3.725	25.0	24.0	290.55	290.15
8	13.952	7.5	7.5	248.60	248.42
9	8.563	110.3	118.4	287.85	288.02
10	11.002	109.6	109.5	289.90	289.94
11	13.186	16.0	16.1	272.53	272.69
12	12.032	67.1	68.1	289.02	289.01

Derived from a 50x50 block of pixels starting at scanline 52393, pixel 200 on 23 June 1992. This data is over clear ocean. Values listed in columns marked (1) are from GSFC MAS data, while columns marked (2) are from Wisconsin MAS data. SNR is signal to noise ratio defined as mean radiance for 50x50 pixel box divided by standard deviation. It should be noted that different spectral response and Planck function information is used at Wisconsin, which accounts for the slight difference in temperature values.

(3) MODIS Level-2 Shell Prototype

Further design work was undertaken on the MODIS Level-2 Shell Prototype. Copies of the design documents produced so far are included overleaf for comment.

MODIS Level-2 Shell Prototype Concept

Liam E. Gumley and J.J. Pan
MODIS Science Data Support Team
25 September 1992

Objective

The MODIS Level-2 Shell Prototype (MLSP) will explore the concepts and techniques to be used in the MODIS Level-2 Shell through the use of data from the MODIS Airborne Simulator (MAS) and representative science algorithms. The emphasis will be on the mechanics of the shell itself i.e.

- control mechanisms
- data flows
- algorithm interactions
- input/output redundancy
- process scheduling
- process efficiency

The prototype is intended to be a simulation testbed for these concepts only, and not a simulation of MODIS science.

Description

The first version of the MLSP will contain the following components:

- Level-1B input data sets from the MAS
- Ancillary input data sets (TBD)
- Land/sea discrimination algorithm
- Cloud/snow discrimination algorithm
- Normalized difference vegetation index (NDVI) algorithm
- Sea surface temperature (SST) algorithm
- Aerosol optical depth (AOD) algorithm
- Level-2 output data sets (archive, metadata, browse)

Future versions of the MLSP will contain more sophisticated science algorithms such as:

- Biome - Biogeochemical model (BGC) algorithm
- Cloud Optical Depth (COD) algorithm
- Cloud Top Temperature/Height (CTT) algorithm

The MLSP will integrate the science algorithms into a hierarchical structure which allows data and control flows between the algorithms in an orderly manner. The shell will be responsible for retrieving the data required by the algorithms from the input Level-1B and ancillary data sets, and for creating Level-2 data products in archive form. Metadata and browse products are then created from these archive products.

Details

The MLSP will be implemented on a Unix system in ANSI-C. The science algorithms will be implemented in either FORTRAN-77 or ANSI-C. The CFORTRAN system will be used to interface C and FORTRAN science algorithms with the controlling shell. The key function of CFORTRAN is the ability to pass parameters of various types between C and FORTRAN. The NetCDF software library will be used for data formatting. No other software tools are required at present.

The input data used by the MLSP will be MAS Level-1B flight lines. These contain calibrated, geolocated MAS radiances and ancillary information and are stored in netCDF format.

Ancillary datasets will be used as required, and will include at least the following items:

- Land/sea topography data base
- Total ozone climatology data base
- Exoatmospheric solar spectral irradiance data base

The science algorithms will be constructed as either subroutines or functions which exchange input and output data as well as control information with the shell. The algorithms will be designed to contain computations only. They will not contain any file input/output calls, apart from those necessary internally to access ancillary data files. The flow of input and output data will be controlled by the shell.

Algorithms

Land/sea discrimination

This will utilize a database of land/sea topography to determine which pixels are over land or sea. Ancillary data required is a topography database of comparable spatial resolution to the MAS data.

Cloud/snow discrimination

This will utilize a visible reflectance threshold test, and a visible/near-infrared channel ratio test to determine the existence of cloud or snow. Ancillary data required is a database of exoatmospheric solar irradiance at MAS wavelengths to determine reflectances.

Normalized difference vegetation index (NDVI)

This will use the well known visible/near-infrared reflectance difference ratio to estimate the NDVI. Ancillary data required is a database of exoatmospheric solar irradiance at MAS wavelengths to determine reflectances. Cloud/snow and land/sea masking are required.

Sea surface temperature (SST)

This will utilize a split window algorithm where two infrared channels are used to estimate the skin temperature. Cloud/snow and land/sea masking are required.

Aerosol optical depth (AOD)

This will utilize a single scattering atmospheric correction to estimate AOD over dark ocean surfaces. Ancillary data required is a database of exoatmospheric solar irradiance at MAS wavelengths to determine reflectances, and a total ozone climatology.

Output products

The MLSP will generate output products of the same type expected by MODIS, including:

- Level-2 archive data
- Metadata
- Browse data

The Level-2 archive data will be stored in NetCDF format. An appropriate structure for the Level-2 output files will be designed and implemented. Utility programs will be developed to image these data files.

Metadata will be extracted from the Level-2 archive data routinely. A list of desired metadata items will be produced, and a utility to extract these items from the archive data will be developed.

Browse data will be generated from the Level-2 archive data routinely. This will consist of imagery at spatially subsampled resolution in a standard image format (e.g. HDF or GIF).

Detailed Algorithm Descriptions

(1) Land/sea discrimination

A database of world topography is used to establish whether a given latitude/longitude is land or sea. Currently a 10 nautical mile resolution database is available, however a database with resolution closer to that of a MAS pixel is being investigated.

(2) Cloud/snow discrimination

Visible and near-infrared reflectances are used to determine whether a given pixel contains cloud or snow. Bright pixels are identified using a reflectance threshold test at around 0.66 microns. Separation of these bright pixels into cloud and snow is done using a reflectance threshold test at around 1.60 microns, where clouds are bright and snow is dark. An additional test may be necessary to check for sunglint regions over water.

(3) Normalized difference vegetation index (NDVI)

This index is calculated by a difference ratio of the form

$$\frac{\text{(near infrared radiance) - (visible radiance)}}{\text{(near infrared radiance) + (visible radiance)}}$$

where the near infrared radiance is at around 0.8-0.9 microns, and the visible radiance is at around 0.67 - 0.7 microns. It may be necessary to convert these radiances to reflectances if longer wavelength infrared (e.g. 1.6 micron) channels are used. This algorithm is only used over land when no cloud or snow is present.

(4) Sea surface temperature (SST)

Two longwave infrared channels are used to compute a SST estimate using the split window algorithm of the form

$$SST = T_{11} + a(T_{11} - T_{12}) + b$$
$$a = 1.4846$$
$$b = 0.0 \text{ (bias correction)}$$

where T_{11} and T_{12} are the 11 micron and 12 micron equivalent black body temperatures in Kelvin. This algorithm was developed at the University of Wisconsin-Madison for the MAMS instrument. This algorithm is only used over water when no cloud is present.

(5) Aerosol optical depth (AOD)

This algorithm uses a single visible channel to estimate the aerosol optical depth over the ocean. It assumes that the ocean surface is essentially dark at wavelengths greater than 0.67 microns. Thus in the absence of sunglint, the radiance received by a high altitude sensor above most of the scattering constituents of the atmosphere may be approximated as

$L_t = L_r + L_a$
L_t = total sensed radiance
L_r = Rayleigh single scattered radiance
L_a = Aerosol single scattered radiance

If the Rayleigh scattering component L_r is estimated with reasonable accuracy then the problem may be rewritten as

$L_a = L_t - L_r$

and it is then necessary to find the aerosol optical depth which gives rise to the aerosol single scattered radiance L_a. This algorithm is only used over dark water in the absence of clouds.

MODIS Level-1 Software Design Status
Thomas E. Goff
1 October, 1992

teg@cheshire.gsfc.nasa.gov,
(301) 982-3704
tgoff on GSFC mail

-- Miscellaneous Status--

- **Replace program enhancements** - A wild card facility was added to the replace program to allow the conversion of UNIX man and more page output to be captured by my PC and printed with character enhancements (bold, italics) on the HP LaserJet. This was required in order to obtain copies of the documentation for the UNIX machines that we use daily.

- **Further Porting of the MODIS sample C programs.** - Both the fdump and replace programs in their final form using ANSI C prototyping guidelines, previously on the SGI, Sun, and PC platforms, have been transferred to the HP730 (handle: modis1). These programs are written in ANSI C and will not compile with the native compilers on the VAX, Sun, or HP machines. Both the Sun and HP computers have the gnu C compiler installed, thereby allowing these programs to be compiled and executed on the Sun and HP. HP's optional C compiler has an ANSI standard mode. The ANSI header files across platforms contain discrepancies that will cause warnings and possible errors when porting code across platforms. These header files will need to be sanitized in the EOS era in order for the posts to be completely successful. ANSI compatibility is still a young technique.

- **HP 9000/730 Capabilities.** - I have generated a memo outlining my requested enhancements to the modis1 computer. I have also set up my account on both the HP and Sun machines with command completion, history, and editing capabilities.

- **Microsoft Project** - This project support tool is continuing to be utilized in the planning of the MODIS Level-A and Level-1B designs. Level-2 design efforts will be added to the suite of existing project files (workspace). Consideration should be given to subscribing to the independent Project Views newsletters for MS Project 3.0 users.

- Futures -

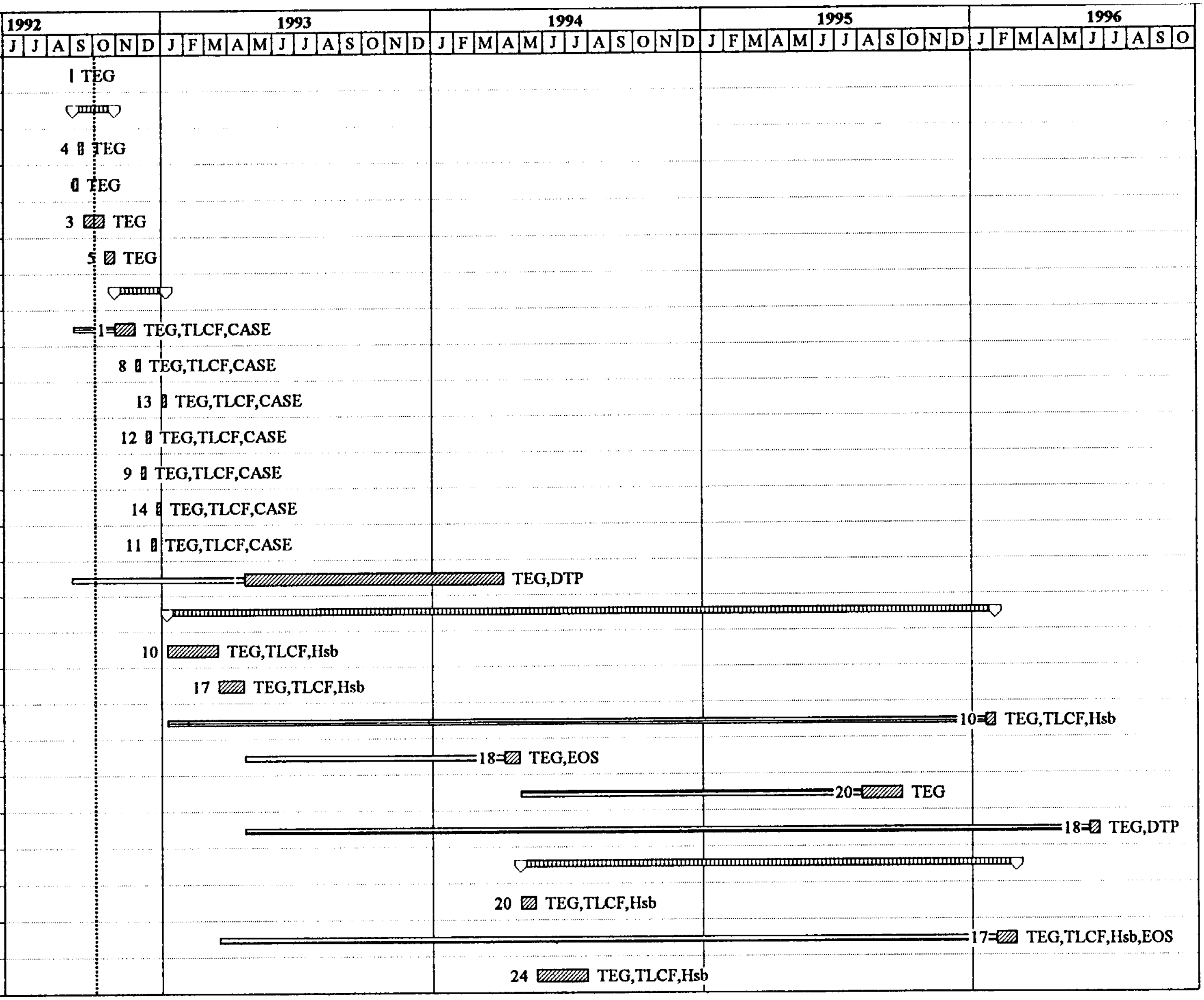
- SLIP and/or ppp is forever being investigated. I have additional names and resources which I am trying to reach.

- Additional software will be loaded onto the modis1 computer to provide debugging, enhanced make, code checking, text processing, postscript conversions, screen dumps, etc. as time allows.

MODIS Data Product Generator Design

MODIS Level-1B - Delay Gantt, 1 Oct '92

ID	Name
1	publish alpha version structure charts
2	preliminary requirements
3	verify DADS interrogation messages
4	determine data product contents
5	determine metadata contents
6	determine cube and granule header contents
7	perform CASE design
8	develop ß version structure charts
9	examine initiation messages
10	generate termination messages
11	handle dynamic status messages
12	create processing log entries
13	generate data flow exception messages
14	setup processing items
15	update/publish assumptions/tracking list
16	MODIS code creation
17	source code generation
18	source code compiling
19	create/update make files
20	perform source code QA
21	source code walkthroughs
22	External Interface Document
23	§ MODIS execution on TLCF
24	code debugging
25	PGS tool kit interfacing
26	execution testing



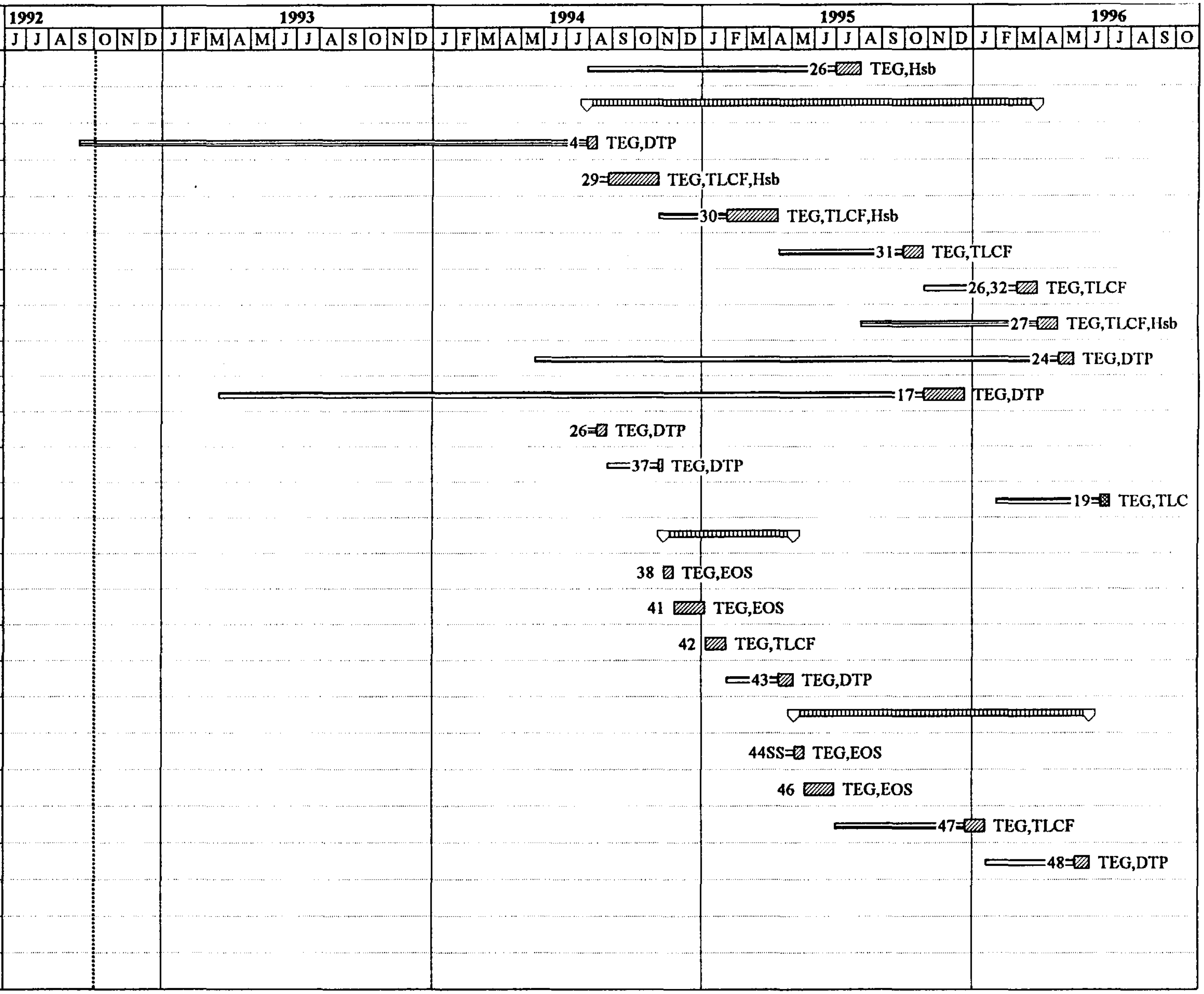
Project: MODIS Level-1B
Date: 1/10/92

Critical	▩	Milestone	◆	Delay	▭
Noncritical	▨	Summary	▽▭▽	Slack	
Progress	▬	Rolled Up	◇		

MODIS Data Product Generator Design

MODIS Level-1B - Delay Gantt, 1 Oct '92

ID	Name
27	performance profiling
28	write data product validator
29	generate data product structure specifications
30	code generation and compiling
31	debugging
32	validation testing
33	data product validation
34	performance on TLCF
35	Software Design Description Document
36	Software Validation and Verification Document
37	Software Installation Guide
38	Software Operations/User's Guide
39	configuration management activities
40	version 1 on PGS
41	port code to PGS
42	execution testing
43	data product validation
44	revise all documentation
45	version 2 on PGS
46	update code on PGS
47	execution testing
48	data product validation
49	revise all documentation
50	
51	
52	



Project: MODIS Level-1B
Date: 1/10/92

Critical ▓ Milestone ◆ Delay ═

Noncritical ▨ Summary ▽ Slack

Progress ■ Rolled Up ◇

DRAFT

MODIS Level-2 Processing Shell Design and Development

J. J. Pan
Research and Data Systems Corp.
(301) 982-3700

Date: September 21 - October 2, 1992

1. C and FORTRAN Interface

Attached are three test programs dealing with the C/FORTRAN interface on the SGI Iris system. The main program of each code is written in C and the subroutines are written in FORTRAN.

In the first C/FORTRAN Interface Demo, which does not include the CFORTRAN tool, examples1 to 5 emphasize the general rules of the interface: character string handling, accessing common blocks of data, array handling, and complex data handling. The second program, which is modified from a FORTRAN demo program developed by Liam Gumley, gives an example of reading a MAS NetCDF file, and the third program is similar to the second program, except using the CFORTRAN tool to handle the interface.

Here are some advantages and disadvantages of using or not using the CFORTRAN tool:

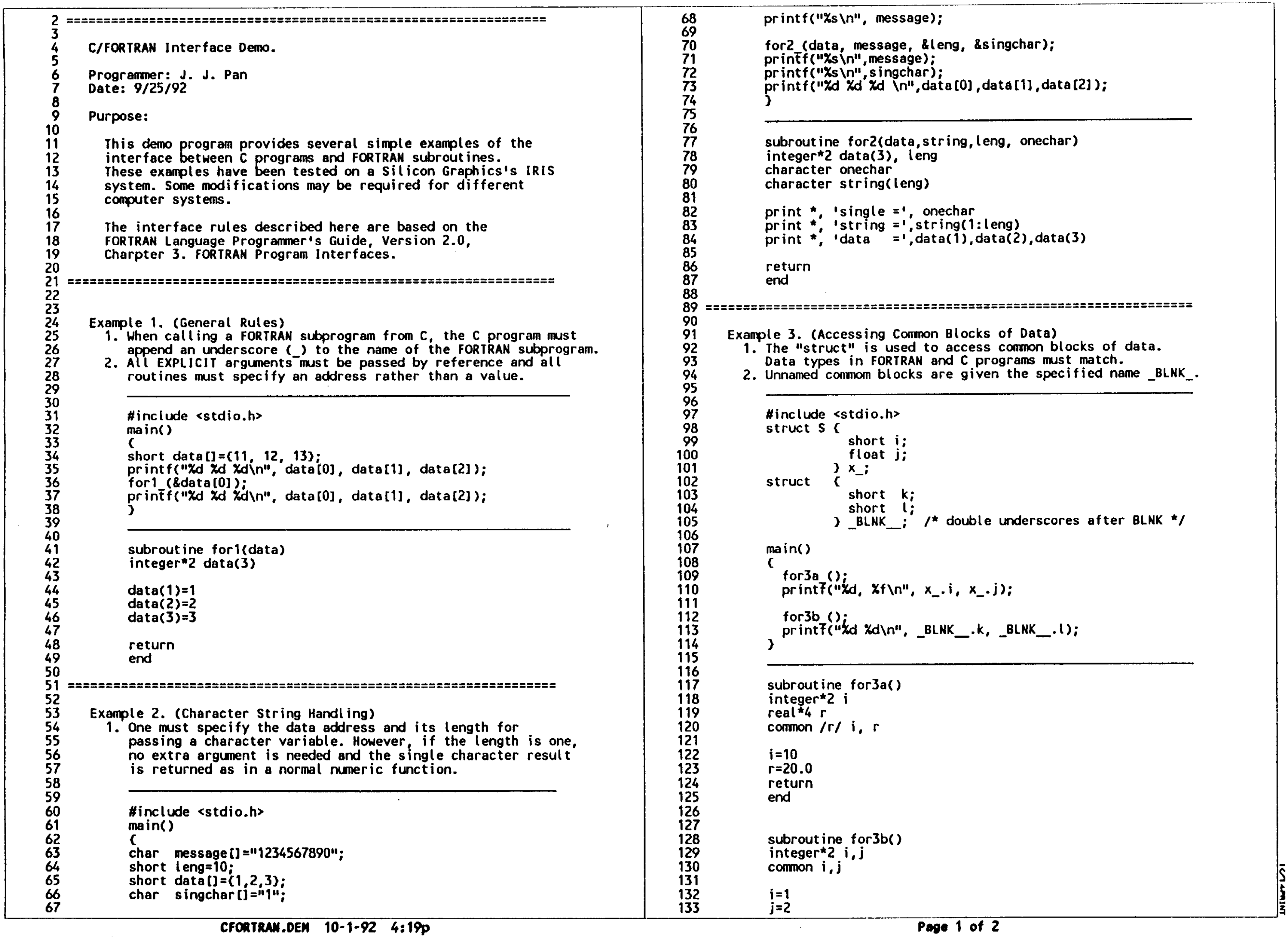
	C/FORTRAN Interface Directly	Using CFORTRAN Tool
Advantages	1. Might get help from the computer dealer. 2. Some documents might be available.	1. It is not necessary to modify FORTRAN subroutines. 2. It is system independent. (for most available systems)
Disadvantages	1. It is system dependent. 2. Might require modifying FORTRAN subroutines which pass char. string.	1. Requires expanding the no. of parameters passed. 2. Reliability and maintenance are to be determined.

2. Shell Prototype Design and Development

The objective of the Level-2 shell prototype has been described in the MODIS Level-2 Shell Prototype Concept (MLSP) report. The major goal in Phase I is to integrate the algorithms. During Phase II we will concentrate on improving the operational capabilities. Some functions of the shell are still under study. They include:

1. Input/Output:
   - How do we retrieve the MAS data and pass it to each algorithm sequentially?
   - How do we keep the flexibility of shell if data processing is done in parallel?
   - How do we store the data products if they are required later in the sequence?
2. Operational Control:
   - How do we control the processing sequence of the shell?
   - How do we handle exceptions such as running one particular algorithm?
   - How do we manage the metadata and browse data generation?
3. Data Flows:
   - How do we determine whether the required input data are available?
   - How do we design a data "log" for tracking the status of data processing?
   - How do we manage data files?

Some additional questions, such as the useage of the PGS tool kit, will be addressed in the future.



```
==============================================================

   C/FORTRAN Interface Demo.

   Programmer: J. J. Pan
   Date: 9/25/92

   Purpose:

     This demo program provides several simple examples of the
     interface between C programs and FORTRAN subroutines.
     These examples have been tested on a Silicon Graphics's IRIS
     system. Some modifications may be required for different
     computer systems.

     The interface rules described here are based on the
     FORTRAN Language Programmer's Guide, Version 2.0,
     Charpter 3. FORTRAN Program Interfaces.

===============================================================


   Example 1. (General Rules)
     1. When calling a FORTRAN subprogram from C, the C program must
        append an underscore (_) to the name of the FORTRAN subprogram.
     2. All EXPLICIT arguments must be passed by reference and all
        routines must specify an address rather than a value.
        __________________________________________________________

        #include <stdio.h>
        main()
        {
        short data[]={11, 12, 13};
        printf("%d %d %d\n", data[0], data[1], data[2]);
        for1_(&data[0]);
        printf("%d %d %d\n", data[0], data[1], data[2]);
        }
        __________________________________________________________

        subroutine for1(data)
        integer*2 data(3)

        data(1)=1
        data(2)=2
        data(3)=3

        return
        end

===============================================================


   Example 2. (Character String Handling)
     1. One must specify the data address and its length for
        passing a character variable. However, if the length is one,
        no extra argument is needed and the single character result
        is returned as in a normal numeric function.
        __________________________________________________________

        #include <stdio.h>
        main()
        {
        char  message[]="1234567890";
        short leng=10;
        short data[]={1,2,3};
        char  singchar[]="1";

        printf("%s\n", message);

        for2_(data, message, &leng, &singchar);
        printf("%s\n",message);
        printf("%s\n",singchar);
        printf("%d %d %d \n",data[0],data[1],data[2]);
        }
        __________________________________________________________

        subroutine for2(data,string,leng, onechar)
        integer*2 data(3), leng
        character onechar
        character string(leng)

        print *, 'single =', onechar
        print *, 'string =',string(1:leng)
        print *, 'data   =',data(1),data(2),data(3)

        return
        end

===============================================================

   Example 3. (Accessing Common Blocks of Data)
     1. The "struct" is used to access common blocks of data.
        Data types in FORTRAN and C programs must match.
    2. Unnamed commom blocks are given the specified name _BLNK_.
        __________________________________________________________

        #include <stdio.h>
        struct S {
                   short i;
                   float j;
                 } x_;
        struct   {
                   short  k;
                   short  l;
                 } _BLNK__;  /* double underscores after BLNK */

        main()
        {
          for3a_();
          printf("%d, %f\n", x_.i, x_.j);

          for3b_();
          printf("%d %d\n", _BLNK__.k, _BLNK__.l);
        }
        __________________________________________________________

        subroutine for3a()
        integer*2 i
        real*4 r
        common /r/ i, r

        i=10
        r=20.0
        return
        end


        subroutine for3b()
        integer*2 i,j
        common i,j

        i=1
        j=2
```

CFORTRAN.DEM 10-1-92 4:19p

Page 1 of 2

```
      return
      end
```

==

Example 4. (Array Handling)

1. FORTRAN stores arrays in Column-major order with the leftmost subscript varying the fastest. C, however, uses Row-major order with the rightmost subscript varying the fastest.

```
#include <stdio.h>
main()
{
short i,j,k;
short n=0;
short p1=1, p2=1, p3=3;
short array[4][3][2];

for (k=0; k<4; k++)
  for (j=0; j<3; j++)
    for (i=0; i<2; i++)
        {  array[k][j][i]=n++;
           printf(" %d %d %d %d\n", i,j,k,array[k][j][i]);
        }

for4_(array, &p1, &p2, &p3);

for (k=0; k<4; k++)
  for (j=0; j<3; j++)
    for (i=0; i<2; i++)
      printf(" %d %d %d %d\n", i,j,k,array[k][j][i]);
}
```

```
      subroutine for4(array, p1, p2, p3)
      integer*2 p1,p2,p3
      integer*2 array(2,3,4)

      do 10 k=1,4
      do 10 j=1,3
      do 10 i=1,2
      print *, i-1,j-1,k-1,array(i,j,k)
   10 continue
      array(p1+1,p2+1,p3+1)=0

      return
      end
```

==

Example 5. (Complex Data Handling)

1. The "struct" is used to pass complex values.

```
#include <stdio.h>
struct {float real,imag;} x;
main()
{
for5_(&x);
printf("%f %f \n", x.real, x.imag);
}
```

```
      subroutine for5(x)
      complex*8 x

      x = cmplx(1.0, 2.0)
      print *, x

      return
      end
```

```
/************************************************************************

       Interface of C and FORTRAN programs on IRIS

       Programmer: J. J. Pan
       Date : 9/28/92

       This program is based on the demo FORTRAN program:
       SIMPLE.F, developed by Liam Gumley, which gives a quick hack
       to demonstrate reading of a MAS netCDF file.

************************************************************************/

/*     include the netcdf.h definitions                                */

#include <stdio.h>
#include "netcdf.h"


main()
{

/*     set up necessary data types for netCDF (note that you should
       leave these as the default type for your compiler)         */

long   cdfid, rcode, dataid;
static long   start[3], count[3];

/*     set up types for variables and attributes                  */

long   vrtype, vrlen, ttype, tlen;
short  i;
short  leng, value[1];
float *scale;
char  *string;

      printf(" Enter Line, Band, and Pixel :\n");
       for (i=0; i<3; i++)
        scanf("%d", &start[i]);

      printf("==%d,%d,%d\n",start[0],start[1],start[2]);


/*      set netCDF error options                        */
        ncopts = NC_VERBOSE | NC_FATAL;

/*      open the netcdf file                            */
        cdfid = ncopen( "test.cdf", NC_NOWRITE);

/*      get the variable id for the desired variable    */
        dataid = ncvarid( cdfid, "CalibratedData");

/*      get the scale factor values (from attribute scale_factor) */
        ncattinq (cdfid, dataid, "scale_factor", &vrtype, &vrlen);
        ncattinq (cdfid, dataid, "units", &ttype, &tlen);
        scale = (float *) malloc(vrlen*nctypelen(vrtype));
        string = (char *) malloc(tlen*nctypelen(ttype));

        ncattget( cdfid, dataid, "scale_factor", (void *)scale);
        printf(" scale=%f, %f, %f\n", scale[0], scale[1],scale[2]);

/*      get the units text description (from attribute units) */
        ncattget( cdfid, dataid, "units", (void *)string);

/*      set the pixel, channel and record counters       */
        count[0] = 1;
        count[1] = 1;
        count[2] = 1;

/*      get the hyperslab of data (one number in this case)    */
        ncvarget( cdfid, dataid, start, count, (void *) value);

        printf(" cdfid =%d\n", cdfid);
        printf(" dataid=%d\n", dataid);
        printf(" start =%d, %d, %d \n",start[0],start[1],start[2]);
        printf(" count =%d, %d, %d \n",count[0],count[1],count[2]);
        printf(" value =%d\n", value[0]);

        leng=29;
/*      calling a Fortran subroutine                               */
        code1_(&value[0], &scale[0], &start[0], string, &leng);
}



        subroutine code1(value, scale, start, string, length)

        integer*4    start(3)
        integer*2    value, length
        real*4       scale(12)
        character*72 string

        print *, 'in code1==  value= ', value
        print *, '        ==  scale= ', scale(1),scale(2),scale(3)
        print *, '        ==  start= ', start(1),start(2),start(3)

c       write the resulting radiance, after rescaling to a real number
c       print *, ' Radiance =', real(value)*scale(start(2))
        print *, string(1:length)

        return
        end
```

```
/*********************************************************************

        Demo of the CFORTRAN tool

        Programmer: J. J. Pan
        Date: 9/24/92

        This program is based on the demo FORTRAN program:
        SIMPLE.F, developed by Liam Gumley, which gives a quick hack
        to demonstrate the reading of a MAS netCDF file.

        The Main program is written in C and the subroutine is in Fortran.
        CFORTRAN is used as the tool to interface C and FORTRAN programs.

*********************************************************************/

/*      include the netcdf.h and cfortran.h definitions              */

#include <stdio.h>
#include "netcdf.h"
#include "cfortran.h"

#define CODE1(A,B,C,D) CCALLSFSUB4(CODE1,code1,SHORT,FLOATV,LONGV, \
         STRING, A, B, C, D)

main()
{

/*      set up necessary data types for netCDF (note that you should
        leave these as the default type for your compiler)           */

long   cdfid, rcode, dataid;
static long   start[3], count[3];

/*      set up types for variables and attributes                    */

long   vrtype, vrlen, ttype, tlen;
short  i;
short  value[1];
float *scale;
char  *string;

       printf(" Enter Line, Band, and Pixel :\n");
        for (i=0; i<3; i++)
         scanf("%d", &start[i]);

       printf("==%d,%d,%d\n",start[0],start[1],start[2]);


/*       set netCDF error options                               */
         ncopts = NC_VERBOSE | NC_FATAL;

/*       open the netcdf file                                   */
         cdfid = ncopen( "test.cdf", NC_NOWRITE);

/*       get the variable id for the desired variable           */
         dataid = ncvarid( cdfid, "CalibratedData");

/*       get the scale factor values (from attribute scale_factor) */
         ncattinq (cdfid, dataid, "scale_factor", &vrtype, &vrlen);
         ncattinq (cdfid, dataid, "units", &ttype, &tlen);
         scale = (float *) malloc(vrlen*nctypelen(vrtype));
         string = (char *) malloc(tlen*nctypelen(ttype));

         ncattget( cdfid, dataid, "scale_factor", (void *)scale);
         printf(" scale=%f, %f, %f\n", scale[0], scale[1],scale[2]);

/*       get the units text description (from attribute units) */
         ncattget( cdfid, dataid, "units", (void *)string);

/*       set the pixel, channel and record counters             */
         count[0] = 1;
         count[1] = 1;
         count[2] = 1;

/*       get the hyperslab of data (one number in this case)    */
         ncvarget( cdfid, dataid, start, count, (void *) value);

         printf(" cdfid =%d\n", cdfid);
         printf(" dataid=%d\n", dataid);
         printf(" start =%d, %d, %d \n",start[0],start[1],start[2]);
         printf(" count =%d, %d, %d \n",count[0],count[1],count[2]);
         printf(" value =%d\n", value[0]);

/*       calling a Fortran subroutine                           */
         CODE1(value[0], scale, start, string);
}




         subroutine code1(value, scale, start, string)

         integer*4    start(3)
         integer*2    value
         real*4       scale(12)
         character*72 string

         print *, 'in code1==  value= ', value
         print *, '        ==  scale= ', scale(1),scale(2),scale(3)
         print *, '        ==  start= ', start(1),start(2),start(3)

c        write the resulting radiance, after rescaling to a real number
c        print *, ' Radiance =', real(value)*scale(start(2))

         print *, string(1:29)

         return
         end
```

MODIS Level 1 Earth Navigation Software Evaluation

Paul A. Hubanks
02 October 1992

I received a group of subroutines and functions that perform earth navigation of satellite pixel data from Fred Nagel (NESDIS, University of Wisconsin) several weeks ago. Some very basic problems have become clear after working with the code. First, the coordinate system conversion routine was not written using structured coding practices. It uses a complex branching logic with multiple exit points. Second, there was an assumption of a spherical earth in the earth navigation routine. Additional code would have to be written to account for the oblateness of the earth and since sections of the code were not "well-structured" it was decided that a major modification of this particular set of routines was not the best option.

The USGS software used for the geolocation of AHVRR data has finally been approved for release. These routines were written in the C language. They have run the code on UNIX, VAX/VMS and SUN workstations implying code portability. The earth navigation routines were developed as a joint effort between the USGS EROS Data Center and the University of Colorado. The software was informally tested before implementation by both facilities, but the test cases and results are "most probably" no longer available. The earth navigation code uses the Clark1866 ellipsoid earth model. The software manager, Doug Hollaren, thought it would be relatively simple to change this to the WGS84 ellipsoid. The software does not include any earth elevation correction. It does, however, correct for time drift of the on-board clock. This drift was on the order of 1/2 second. Operationally, geolocation errors were found to be less that 5 km. Most of this error could be attributed to incorrect ephemeris. The code will be ported to my account on the LTP/VAX computer either today (Friday) or Monday.

I have also been assigned the task to begin collecting preliminary Science Team algorithms for examination. I spoke with Yoram Kaufman and he agreed to release his Aerosol Optical Depth code (product # 2293) after his programmer "cleaned it up". I also have a meeting set up with Si Chee Tsay (Mike Kings programmer) to acquire selected algorithms currently running on MAS data.